Fundamentals of Artificial Intelligence

Prof. Shyamanta M Hazarika
Department of Mechanical Engineering
Indian Institute of Technology - Guwahati

Module - 3
Lecture - 9
Games: Minimax and Alpha-Beta

Welcome to fundamentals of artificial intelligence. Today we will look at the minimax algorithm and alpha-beta pruning.

(Refer Slide Time: 00:41)

Fundamentals of Artificial Intelligence

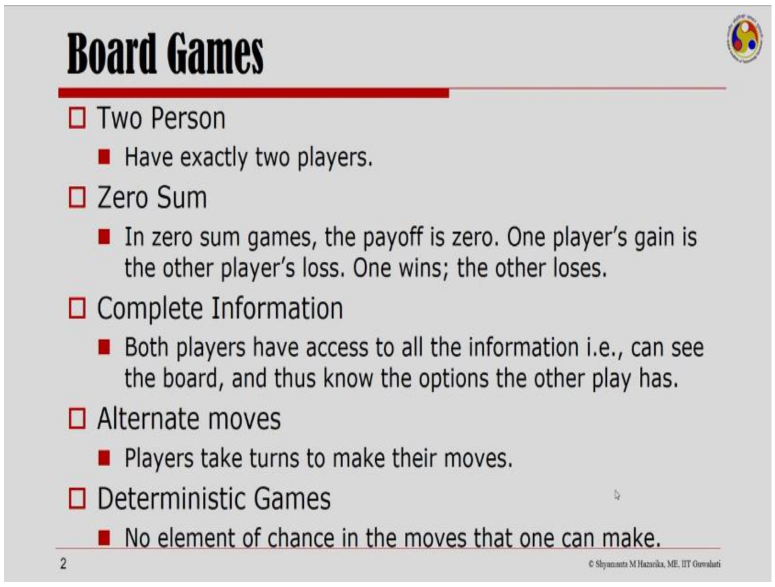
Games: Minimax and Alpha-Beta

Shyamanta M Hazarika
Mechanical Engineering
Indian Institute of Technology Guwahati
s.m.hazarika@iitg.ac.in

http://www.iitg.ac.in/s.m.hazarika/

The minimax algorithm is used to extract the best-first move in a game tree. Alpha-beta is a layering on top of minimax that drastically reduces the search space. We will first look at minimax and see how alpha-beta pruning is applied to reduce search space. We are dealing with board games, games such as chess or checkers or games such as tic-tac-toe.

(Refer Slide Time: 01:14)



Board Games

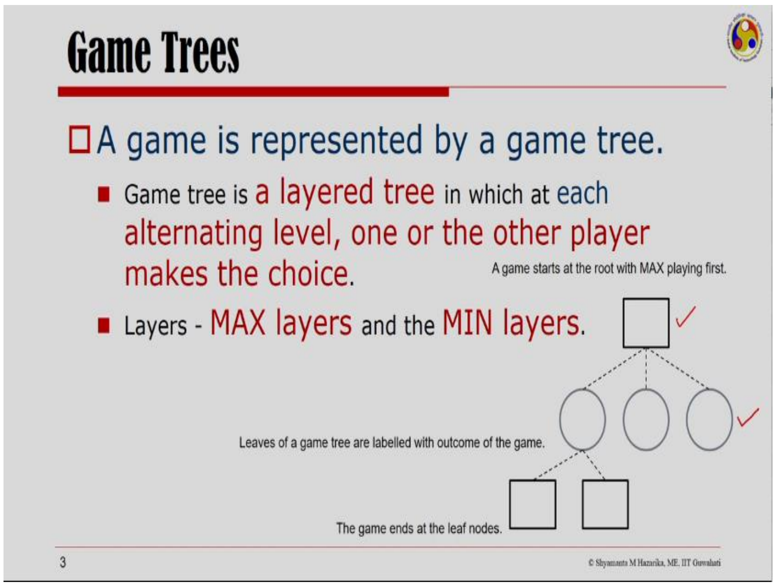
- Two Person
  - Have exactly two players.
- Zero Sum
  - In zero sum games, the payoff is zero. One player's gain is the other player's loss. One wins; the other loses.
- Complete Information
  - Both players have access to all the information i.e., can see the board, and thus know the options the other play has.
- Alternate moves
  - Players take turns to make their moves.
- Deterministic Games
  - No element of chance in the moves that one can make.

2 © Shyamanta M Hazarika, ME, IIT Guwahati

And these games have the following characteristics. One, we are dealing with two-player games; games that have exactly 2 players. These games are called zero sum games. Zero sum games are those in which, if one player wins, the other loses the game. That is, the total payoff is 0. One player's game is other player's loss. The third characteristic of such games is complete information.

Both the players are assumed to have access to all information. That is, can see the board and thus know the options the other player has. These games are alternate move games. The player take turns to make their moves. And finally, we consider these games to be deterministic. There is no element of chance in the moves that one can make.

(Refer Slide Time: 02:27)



Game Trees

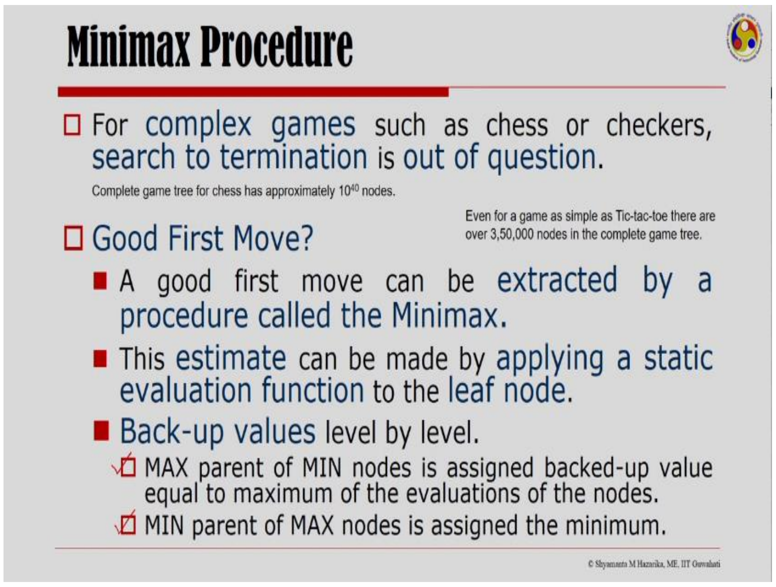
- A game is represented by a game tree.
  - Game tree is a layered tree in which at each alternating level, one or the other player makes the choice.
  - Layers - MAX layers and the MIN layers.

A game starts at the root with MAX playing first.
Leaves of a game tree are labelled with outcome of the game.
The game ends at the leaf nodes.

3 © Shyamanta M Hazarika, ME, IIT Guwahati

We then define a game tree which is used to represent the game. A game tree is a layer tree in which at alternating layer, one or the other player makes the choice. So, we have the Max layer and the Min layer. So, once we have the Max layer; and then, we have the Min layer. And this is how, one after another, the layers of the tree comes. A game starts at the root with the Max layer playing first. And the leaves of the game tree are labeled with outcome of the game. The game ends at the leaf nodes.

(Refer Slide Time: 03:18)



Minimax Procedure

- For complex games such as chess or checkers, search to termination is out of question.
  Complete game tree for chess has approximately 10^{40} nodes.
- Good First Move?
  Even for a game as simple as Tic-tac-toe there are over 3,50,000 nodes in the complete game tree.
  - A good first move can be extracted by a procedure called the Minimax.
  - This estimate can be made by applying a static evaluation function to the leaf node.
  - Back-up values level by level.
    - MAX parent of MIN nodes is assigned backed-up value equal to maximum of the evaluations of the nodes.
    - MIN parent of MAX nodes is assigned the minimum.

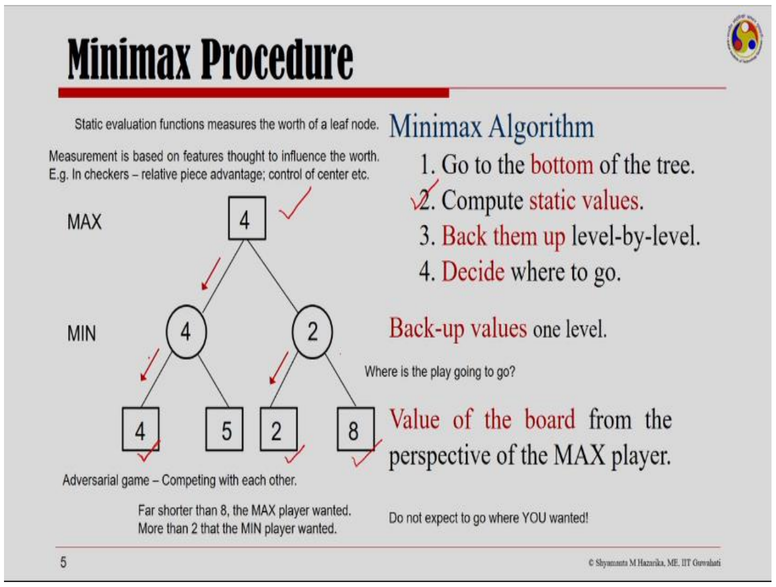
© Shyamanta M Hazarika, ME, IIT Guwahati

Now, let us look at how one gets to know the first best move in such a board game. It is important to realize at this point of time that, for many complex games such as chess, checkers, search to termination is out of question. Like, the complete game of chess has approximately 10 to the power 40 nodes. Given a simple game like tic-tac-toe, has 3,50,000 nodes in the complete game tree.

Therefore, the question is, how we can get to a good first move. The minimax algorithm that we will discuss today is one that can extract a good move. So, this estimate is made by applying a static evaluation function to the leaf nodes. And then, the values are backed up level by level. So, we have the Max parent of a Min node. And then, that parent is backed up values equal to the maximum of the evaluation of the nodes.

This is because we expect the Max player to maximize his game. The Min parent of the Max nodes is assigned to the minimum, because the Min player is with an idea to get to the minimum of the boards.

(Refer Slide Time: 05:08)



Minimax Procedure

Static evaluation functions measures the worth of a leaf node.

Measurement is based on features thought to influence the worth. E.g. In checkers – relative piece advantage; control of center etc.

MAX
MIN
4
4
2
4
5
2
8

Adversarial game – Competing with each other.

Far shorter than 8, the MAX player wanted.
More than 2 that the MIN player wanted.

Minimax Algorithm

1. Go to the bottom of the tree.
2. Compute static values.
3. Back them up level-by-level.
4. Decide where to go.

Back-up values one level.

Where is the play going to go?

Value of the board from the perspective of the MAX player.

Do not expect to go where YOU wanted!

5

© Shyamanta M Hazarika, ME, IIT Guwahati

So, let us take a portion of game tree and try to understand the minmax procedure. So, here is the portion of a game tree. And let us put the evaluations for the leaf nodes. These values 4, 5, 2, 8; these are value of the board from the perspective of the Max player. Now, where is the game going? or Which of the 2 choices does this Max player make? If you look at the game tree, it is not very obvious at the first look.

But if you look it closely and try to see what could Min do best, under such a scenario where he knows the value of the leaf nodes which are actually the Max leaf nodes. So, if you look at this node of the tree that Min is playing, Min would rather go to the left; here; then here. Because, going here would give him a 4, whereas going here would give him a 5. And his idea is to keep the score as low as possible.

So, he goes there and he backups this node value as 4. Now, going by that same logic, if you look at this other node here, Min has 2 choices. Either he plays towards this board or he plays towards this board. Playing towards the left would give him the minimum that he is seeking. And therefore, he will go to the left. And this node that he will backup is with 2. Now, these are the backup values one level up.

Now, having got these backup values, it becomes easier at this level where Max is to realize where to go. Because Max idea is to maximize the total. So, he rather goes towards the left and takes a 4. And therefore, we have a 4 at the root node. Now, a couple of things to observe

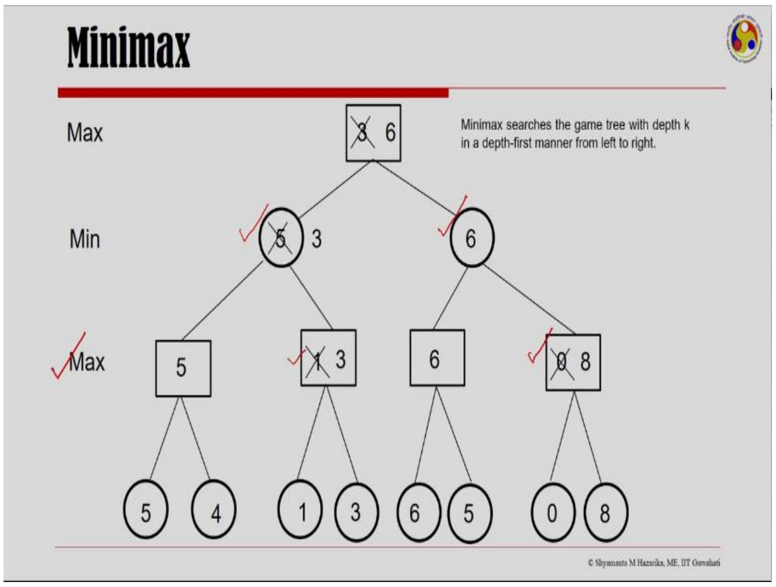
in this small game tree and the way these values are being backed up is in order. Number 1, we should just get to the idea of what the minmax algorithm is.
What did we do? We went to the bottom of the tree; we computed the static values using some evaluation function; and then, backed them up level by level. And at one level, we were at the root. At that point we were able to decide where to go. So, we extracted the best move. Now, one important concept that needs to be emphasized here is the concept of computing the static values.

Static evaluation functions are used to measure the worth of a leaf node. And these measurements are based on features that are thought to influence the worth of that particular node. Like in checkers, it could be the relative piece advantage or who controls the center. In tic-tac-toe it could be, how many ways Max or Min as the case may be, can still finish the game.

These things then give me some idea of what are these nodes worth and these values are then backed up. Another thing that is need to be noticed at this point of time is that we have 8 here, which is the Maximum value possible. And ideally, Max would have loved to arrive at that node. But then, this was not possible. All he arrived at was the value 4. On the other hand, for Min, the minimum value that was possible in this small game tree was 2.

Whereas he was only able to arrive at 4 and not go below that. Now, this is because these are adversarial games. Min and Max are competing against each other. And you do not expect to go where you wanted, because the competitor is always trying to stop you from going there. And that is why we arrived at 4 which was far shorter than 8 and more than 2, what the Min player would have wanted.

(Refer Slide Time: 10:34)



Minimax

Minimax searches the game tree with depth k in a depth-first manner from left to right.

This minmax search is actually a depth-first search. The minmax search searches the game tree with depth k in a depth-first manner from left to right. So, let us try and illustrate this idea. Hope this works. And we can see how the depth-first search is done by the minmax procedure. So, here is a game tree and all of its leaf nodes are marked. So, let us now see how minmax searches the game tree with depth k or in a depth-first manner from left to right.

So, it starts at the max node, comes down 1 level which is the level of the Min player. It comes to the Max layer, 1 level down; and finally arrives at the leaf node. Thereafter, it gets the value of the leaf node which is marked 5 at that point. So, that is backed up 1 level. And we get 5 there. Next, this is a depth-first search. So, it has to go to the Min level, to the leaf node one more time. Picks up the value 4.

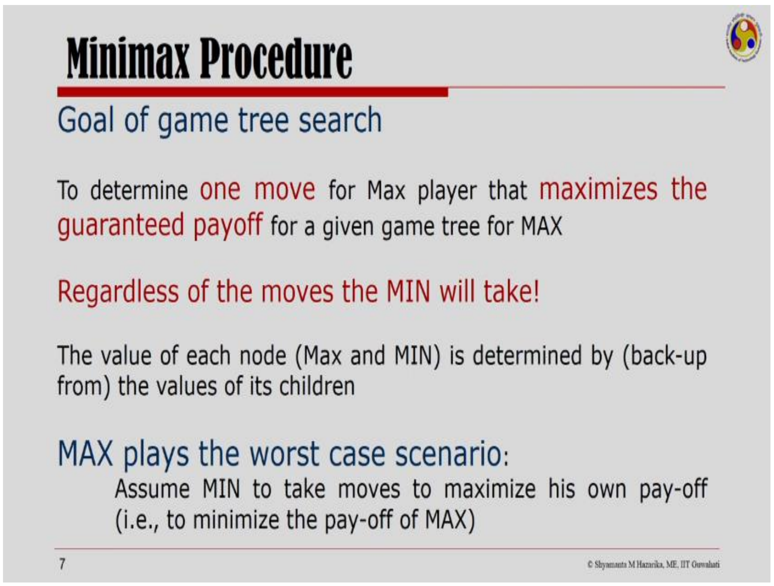
But then, when it backs it up, one got to remember that this is a Max layer. I am expecting the maximum of the 2. 4 is not entered. We still have 5 there, because 4 is less than 5. We work with the maximum, because this is Max. So, this is backed one more time up. And Min has a 5 there. Now, one level down, it goes there, comes back to the leaf node. And 1 is backed up. So, we have a 1.

And then, it visits 3. Now, this is interesting. Because, now we have a leaf node which is more than the value that is at the Max level. I have 1 here, whereas now, the leaf node value is 3. So, it overwrites 1 and keeps the new maximum which is 3. Backs it up 1 more level.

And this is again interesting to note. Because this is a Min level now and I have 5 there; whereas the backed up value is 3. And Min would prefer 3 rather than 5.
So, 5 is overwritten and 3 is kept. And finally, 3 is backed up to the root of the tree. This is how the left tree is completed. Now, to the right subtree, it goes and visits the leaf node, backs up the value 6 and goes and visits the leaf node which is with value 5. So, even if it comes back, it does not override. Because Max, we are expecting it to have 6 rather than 5. It backs up 6 to the Min and then comes down 1 final time to the leaf node which is 0; backs up the 0 value.

But interesting to note at this point of time that, again we have a Max node and the leaf value is 8. So, 0 would be overwritten by 8. Whereas, now when I want to backup 8, I have a 6. And this is a Min node. So, between 6 and 8, here 6 is preferred, because I am looking for the minimum value of the 2 values. So finally, 6 is backed up. And the value of that final root node is 6. So, the total backed up value of this game is 6 there.

(Refer Slide Time: 15:21)



Minimax Procedure

Goal of game tree search

To determine one move for Max player that maximizes the guaranteed payoff for a given game tree for MAX

Regardless of the moves the MIN will take!

The value of each node (Max and MIN) is determined by (back-up from) the values of its children

MAX plays the worst case scenario:

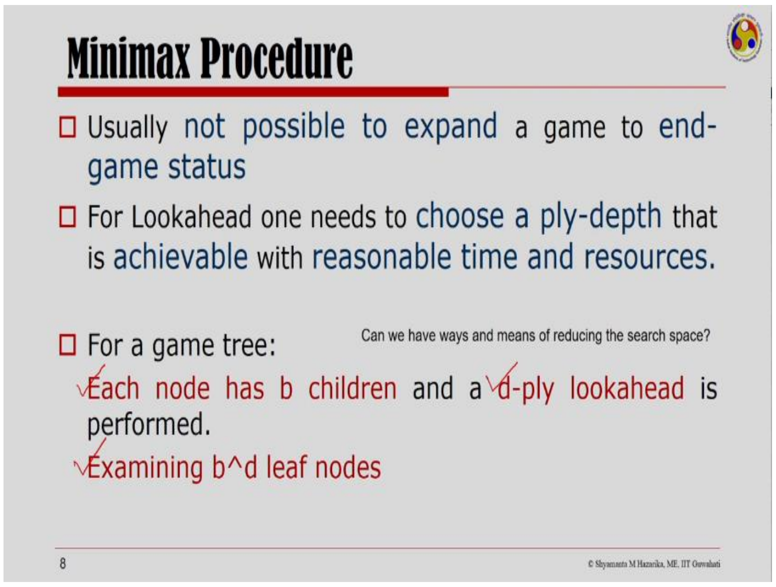
Assume MIN to take moves to maximize his own pay-off (i.e., to minimize the pay-off of MAX)

7

© Shyamanta M Hazarika, ME, IIT Guwahati

Now, we did determine the best move for the Max player. And this guaranteed somehow that the payoff was maximum for him, regardless of what the move the Min will take. The value of each node is determined by the values of its children. And Max plays the worst case scenario. That is, he assumes when he is doing minimax is that Min takes moves to maximize his own payoff. That is to minimize the payoff of Max.

(Refer Slide Time: 16:01)



Minimax Procedure

- Usually not possible to expand a game to end-game status
- For Lookahead one needs to choose a ply-depth that is achievable with reasonable time and resources.
- For a game tree:

Can we have ways and means of reducing the search space?

✓Each node has b children and a ✓d-ply lookahead is performed.

✓Examining b^d leaf nodes

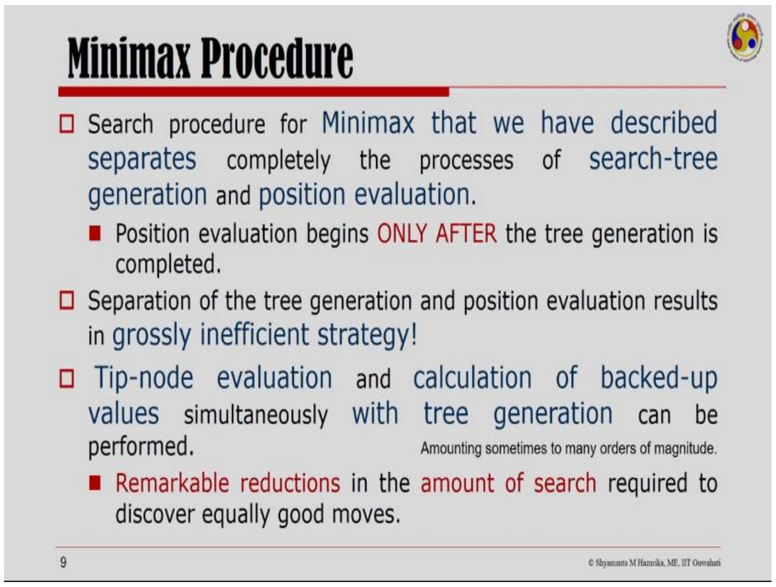
8

© Shyamanta M Hazarika, ME, IIT Guwahati

As already pointed out, it is not possible to expand a game to the end game status. For lookahead, one needs to choose, therefore a ply-depth that is achievable with reasonable time and resource. Now, we need to decide how much depth to we will go when we were, we are searching for the best-first move. This is because of the following reason. For a game tree, each node has b children let us say and we are doing a d-ply lookahead.

That means, we are trying to go to some level d. That would require that we examined b to the power d leaf nodes. Can we have ways and means of reducing this, because this is a huge search space? So, can we somehow reduce this space?

(Refer Slide Time: 17:06)



Minimax Procedure

- Search procedure for Minimax that we have described separates completely the processes of search-tree generation and position evaluation.
  - Position evaluation begins ONLY AFTER the tree generation is completed.
- Separation of the tree generation and position evaluation results in grossly inefficient strategy!
- Tip-node evaluation and calculation of backed-up values simultaneously with tree generation can be performed.

Amounting sometimes to many orders of magnitude.

  - Remarkable reductions in the amount of search required to discover equally good moves.

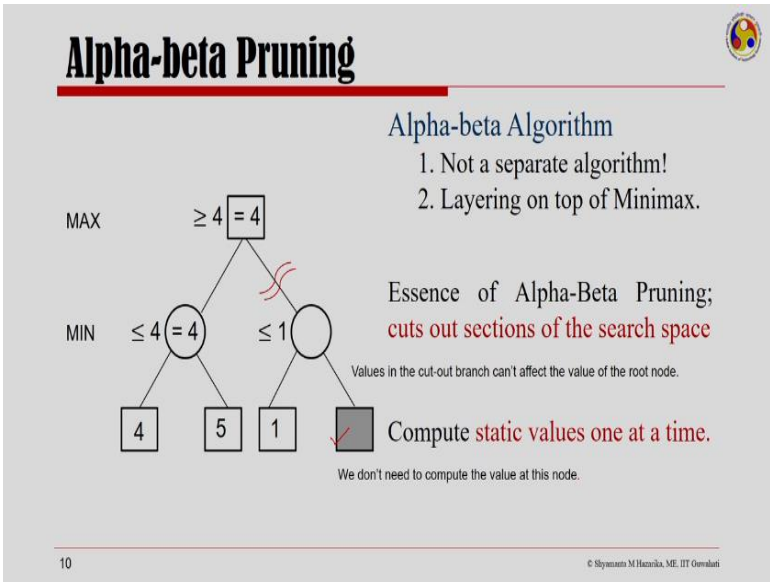
9

© Shyamanta M Hazarika, ME, IIT Guwahati

Now, search procedures for minmax that we have described up till now separates completely the process of search tree generation and position evaluation. The position evaluation was done only after the tree generation is completed. Separation of tree generation and position evaluation results in grossly inefficient strategy. So, is there some way in which tip-node evaluation and calculation of backed up values simultaneously with tree generation can be done?

If this could be done, we could have remarkable reductions in the amount of search required to discover equally good nodes. So, this is exactly what we look for in this layering on minimax that we are talking of. And that amounts sometimes to many orders of magnitude reduction in the amount of search.

(Refer Slide Time: 18:11)



Alpha-beta Pruning

Alpha-beta Algorithm
1. Not a separate algorithm!
2. Layering on top of Minimax.

Essence of Alpha-Beta Pruning; cuts out sections of the search space

Values in the cut-out branch can't affect the value of the root node.

Compute static values one at a time.

We don't need to compute the value at this node.

10

© Shyamanta M Hazarika, ME, IIT Guwahati

That is what is alpha-beta pruning. So, let us try to understand alpha-beta pruning by taking the same game tree that we had used to illustrate the minimax procedure. But then, one thing to notice at this point of time is that we do not evaluate all the leaf nodes. So, we start by computing the static values one at a time. We compute the first static value which is 4. Now, immediately we have a realization that 1 level up, that is the Min, any value here that he would be looking for would be less than or equal to 4.

We then compute the next value and realize that this value is 5. And therefore, Min can stay with 4. Now, 1 level up when we want to backup. A realization comes that if Max had taken this path, then he would have been at least guaranteed 4. That means, at this point of time, Max can have values which would be greater than or equal to 4. We compute the third static value which is 1.

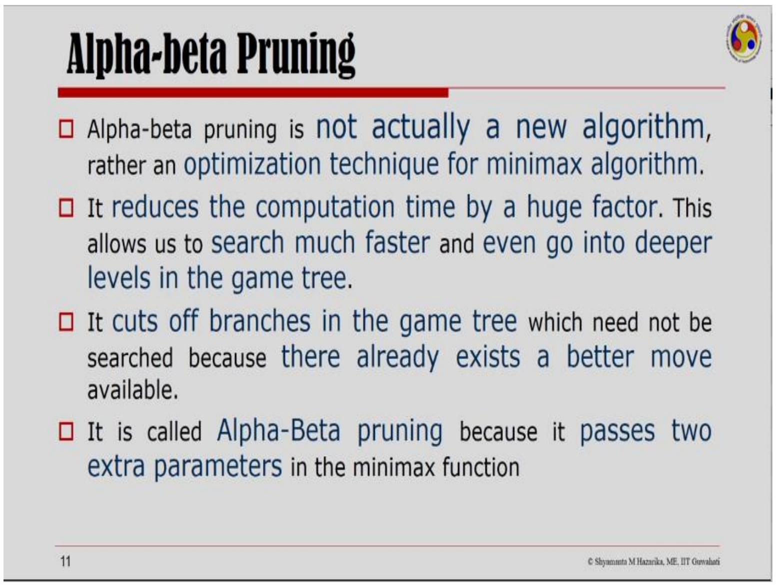
At that point of time, we can see that the Min value now would be something which is less than or equal to 1. This is where a little introspection will make us realize that this side on the left, if Max wanted to go, you would have at least 4. 4 or something greater than 4 is what his current value is. Whereas, if he prefers to come to the right, all he is guaranteed is something less than or equal to 1.

And therefore, he would never prefer to come this way. That is, as good as saying that this side of the tree does not exist for him. So, he can think of throwing away all of this, because as if this tree does not exist. Why is it that this tree does not exist for him? This is because, if Max follows this path; let me repeat this. If Max follows this path, he will have 4. And if Max comes on this path, all he is guarantee to have is something which is less than or equal to 1.

Therefore, this tree does not exist for him. And once he can cut this tree out, he can update that value to be really 4. So, we have got to the same result as in the minimax procedure. But then, we have not evaluated even this node. So, these values that we have cut out, these cannot affect the value of the root node. So, the essence of alpha-beta pruning is that the cut out sections of the search space, they do not affect the value of the root node.

And we therefore do not need to compute these values which have been cut out. So, alpha-beta pruning, we need to realize this, that it is not a separate algorithm. Alpha-beta pruning is actually a layering on top of minimax. It is an optimization procedure on top of minimax to reduce search space and somehow get to the same value that we have got in the minimax procedure.

(Refer Slide Time: 22:28)



Alpha-beta Pruning

- Alpha-beta pruning is not actually a new algorithm, rather an optimization technique for minimax algorithm.
- It reduces the computation time by a huge factor. This allows us to search much faster and even go into deeper levels in the game tree.
- It cuts off branches in the game tree which need not be searched because there already exists a better move available.
- It is called Alpha-Beta pruning because it passes two extra parameters in the minimax function

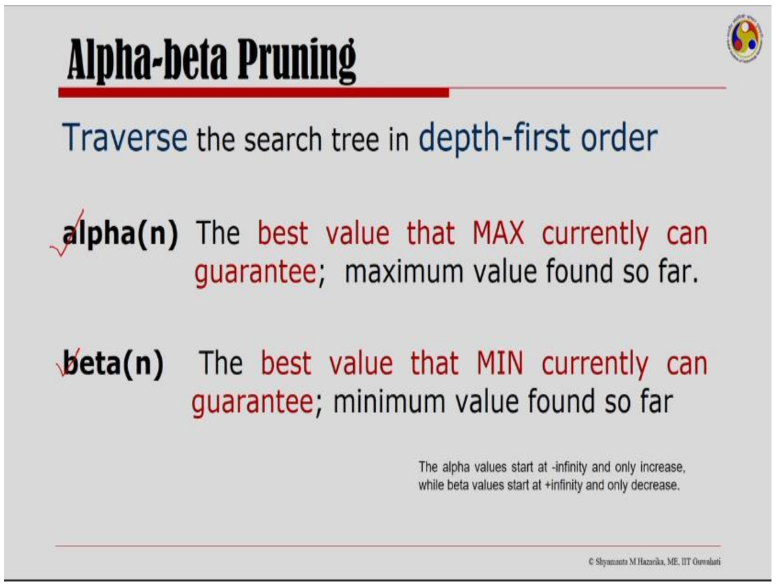
11

© Shyamanta M Hazarika, ME, IIT Guwahati

Alpha-beta pruning therefore reduces the computation time by a huge factor. This allows search much faster. And then, we can even go deeper into the game tree. That is, if we have thought of only computing up to level 4, alpha-beta pruning would allow us to compute to the double of that value. It cuts out branches in the game tree, because these need not be searched. There already exist a better move.

Now, it is called alpha-beta pruning because it passes 2 parameters in the minmax function. One called the alpha and the other called the beta. We will now focus on what is alpha, what is beta, what is an alpha cutoff and what is a beta cutoff.

(Refer Slide Time: 23:22)



Alpha-beta Pruning

Traverse the search tree in depth-first order

alpha(n) The best value that MAX currently can guarantee; maximum value found so far.

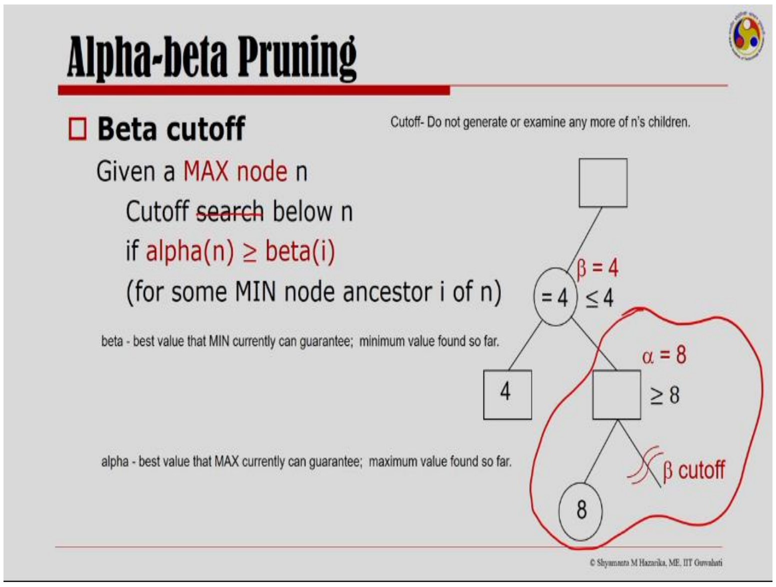
beta(n) The best value that MIN currently can guarantee; minimum value found so far

The alpha values start at -infinity and only increase, while beta values start at +infinity and only decrease.

© Shyamanta M Hazarika, ME, IIT Guwahati

When we are doing minimax procedure, the minimax traverses the search tree in depth-first order. That is what we have seen. Now, the alpha of a particular node n is the best value that Max currently can guarantee. So, this is the maximum value found so far. Whereas beta n is the best value that Min currently can guarantee. So, it is the minimum value found so far. Now, having defined alpha and beta, let us see when we cut out branches of a tree which is called an alpha cut out and which of the cut outs are called the beta cut out.

(Refer Slide Time: 24:09)



Alpha-beta Pruning

- **Beta cutoff**

  Given a MAX node n

  Cutoff ~~search~~ below n

  if alpha(n) ≥ beta(i)

  (for some MIN node ancestor i of n)

Cutoff- Do not generate or examine any more of n's children.

beta - best value that MIN currently can guarantee; minimum value found so far.

alpha - best value that MAX currently can guarantee; maximum value found so far.

© Shyamanta M Hazarika, ME, IIT Guwahati

So, in order to define beta cutoff, let us take a sample tree. Now, let us assume we had a max node n and cutoff search below n is not required. Cutoff below n is possible if alpha n is greater than or equal to beta i. Here, by cutoff, we mean that, we do not generate or examine any more of its children. So, let us take some values for the leaf nodes as shown, 4 and 8 there.

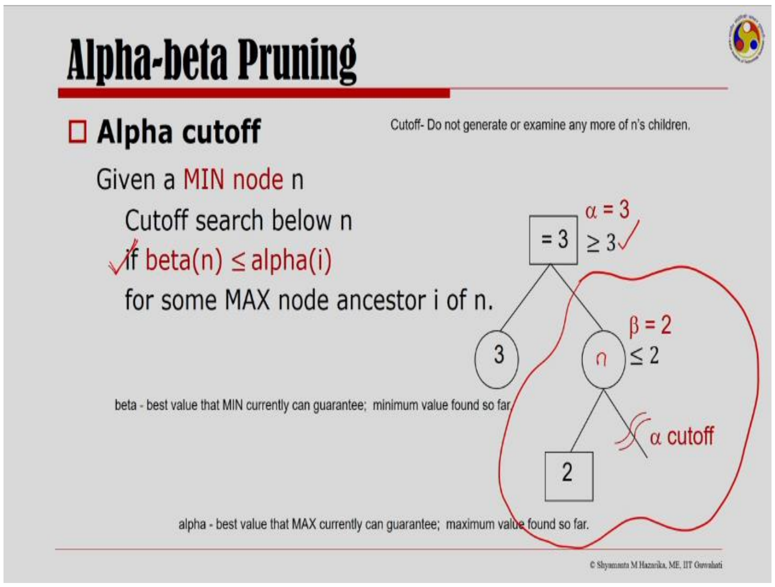
Now, because this is a Min node and because this is a Max node; so, the values that will be possible is: This node will be greater than or equal to 8, whereas this one will be less than or equal to 4. Now, recall what we meant by an alpha and a beta. The alpha is the best value the Max currently can guarantee. So, it is the maximum value found so far. So, at this node, we have alpha equal to 8.

Beta is the best value that Min currently can guarantee. So, beta is the minimum value found so far. Now, let us look at this condition given here. So, I have my node n here. And the alpha value of n is 8. Whereas, the beta value of i which is an ancestor of n is 4. So, the alpha of n which is 8 is greater than or equal to the beta of i which is 4. The condition is satisfied. That means, we do not need to search anything beyond this line.

And this is called a beta cutoff. Now, why is that we do not need to search anything beyond this point? It is interesting to take one minute and note it. Here, we are guaranteed to have something which is greater than or equal to 8. And given that guarantee, here we have something which is guaranteed to keep me to the maximum of 4. Because this is a Min node; so, it will only look for 4 or something less than 4.

So, that means, there is no point of looking for a better solution in this subtree. And that is what is a beta cutoff. Next, we will look at what we mean by an alpha cutoff.

(Refer Slide Time: 27:17)



Alpha-beta Pruning

- Alpha cutoff

  Given a MIN node n

  Cutoff search below n

  if beta(n) ≤ alpha(i)

  for some MAX node ancestor i of n.

Cutoff- Do not generate or examine any more of n's children.

beta - best value that MIN currently can guarantee; minimum value found so far.

alpha - best value that MAX currently can guarantee; maximum value found so far.

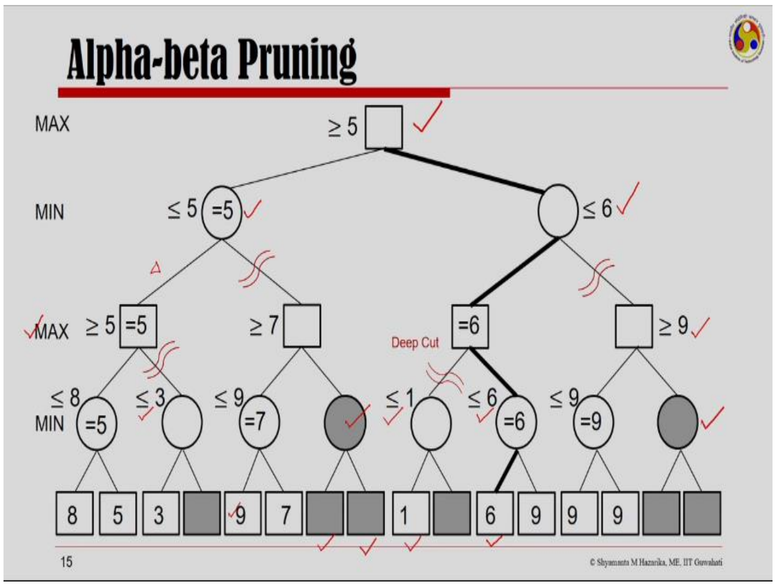
© Shyamanta M Hazarika, ME, IIT Guwahati

So, given a mean node n, we would love to cutoff search below n, when the beta is less than equal to alpha of i. Let us take a small portion of a game tree to understand this. And let us mark the leaf nodes, 3 and 2. Now, this is a Min node. So, it is guaranteed that it will have something which is less than or equal to 2. This is a Max node. It will have something which is guaranteed to be more than or equal to 3.

So, that is greater than equal to 3, that is less than equal to 2. Understanding such a scenario, if I am looking for coming along this path, then I can only have things which is maximum of 2. So, for Max here, there is no point coming this path, as if this does not exist for him. Because if he takes the left side, he will get 3. If he somehow takes the right path, all he can get is a maximum of 2.

So, there is no point looking at this portion of the tree. And this cutoff is called a alpha cutoff. Now, to check the condition, recall that alpha is the best value that Max can have. So, we have an alpha of 3. Beta is the best value that Min can currently have. So, we have a beta of 2. And now, if you compare the beta value at the nth node, which is 2 and which is less than the alpha value of its i ancestor, we can have this cutoff. And this condition is satisfied. So, such a cutoff is called an alpha cutoff.

(Refer Slide Time: 29:39)



Next, let us take a more complex game tree and try to understand alpha-beta pruning. Taking a complex tree and understanding it is important, because there are few things that does not come out in a very simple example that I have been showing. So, let us focus our attention on this tree here. All its leaf nodes are marked. And let us start with the Min node here. So, if you look the Min node there, I have 8 and 5.

Min expects that the minimum should come. And I know 5 should be there. But then, I am doing a depth-first search and I am doing alpha-beta. So here, I will first give Min the value that it could be something which is less than or equal to 8. The moment I see the next leaf node which is five, I revise that value and say that it is not 8 but 5. I back that value up, one more time.

And now, at the Max level here, I can write that it should be greater than or equal to 5. Thereafter, look at this portion. Here we have the leaf node 3. So, Min should be less than or equal to 3. Once I realize I have here, less than or equal to 3 and this is a Max node which realizes that if it moves this path, it will at least have 5. So, it will not love to come this side. So, as if this whole subtree does not exist for him.

So, it will have a cut out here. And once it has a cut out there, we cutoff this portion. You need to realize that I will no longer evaluate this. And therefore, I no longer need to look at that node. I backup the Max value to the Min level. And before that, I need to have a realization that because there is no right subtree; so, this value can be 5. I back that value up. And because this is a Min node, I would have a value which is less than or equal to 5.

I come down at this portion now. This is a Min node. So, its value is less than or equal to 9. And then, when I see 7, I know that this has to be overwritten and I should get to the value 7. So, the correct evaluation of the node is 7. This 7 is backed up and I have the Max value which is greater than equal to 7. And now, there is a interesting realization, which is that if Min comes this side, it would have values which is 5.

But if it follows this, a rise at this node, then all Min will have, are values which are greater than or equal to 7. Now, the idea for Min is to minimize. And therefore, if I have a subtree which guarantees that the value is going to be at least 7 or more; then, when I am searching as Min, this subtree is as good as non-existent for Min. So, we will have a cutoff there. Now, when we have a cutoff at that point, what we no longer need to evaluate is, not only the leaf nodes that I have marked, this one and this one here; but also a generic function which is I do not need to generate the children of this particular Min node.

And that is an amount of saving that alpha-beta pruning allows. Not only reducing evaluation of the leaf nodes, but also reducing generic functions. Now, once we have that value and this right side of the subtree is not there, so we can backup this value here as 5. And once that value is 5, the Max node has a value here, which is greater than or equal to 5. Now, let us focus our attention on the right subtree.

At the root I have 5 coming from the left side. And I start with the leaf node here which is 1. Back it up to the Min level. So, I have something which is less than or equal to 1. Now, this is where it becomes very interesting. And this one would not have been visible in a small tree. Here I have a root node occupied by Max, where if I visit the left tree, I would have 5. And I am expecting something greater than 5 by visiting this side of the tree, the right side of the tree.

Now, when I realize that this tree here, all it will give me is Maximum of 1. I can cut this tree out. One difference that you should notice now, between the earlier cut outs and this cut out is the fact that earlier cut outs were done on adjacent levels. This cut out is called a deep cut. Because, I have gone at least 2 levels down the line to cut out a subtree that I will not explore any further.

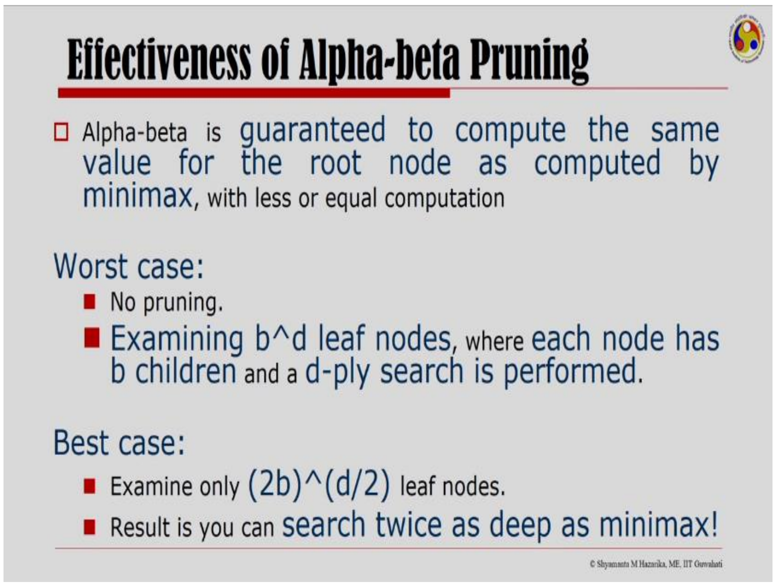
Taking the discussion forward, if you now focus on 6 here, the Min is guaranteed to have less than equal to 6. The next node is 9. But this is not going to satisfy this upper bound that Min has put. So, all I have is 6 here. This side of the tree which have been cut out does not exist for Max. So, Max would backup this value as 6. And now, focusing attention on the Min node and Min will guarantee less than equal to 6 at this node.

I have 9 there. So, I have a Min node. So, at least 9 less than equal to 9. So, I have a 9 marked here. And then, this backs up to the Max node which is saying that it should be greater than or equal to 9. So, when I have a Max node here which is greater than or equal to 9 and I have a Min node which is saying it is less than or equal to 6; then, Min will rather go here and never come on this path. So, for Min, as if this portion of the subtree does not exist.

And we can cut it out. So, we can have a cutoff here. Once we have a cutoff here, these leaves 3, 4 and these generation of children for the Min at this point is not required. And that is the amount of saving. So, this is what we have at the Min node now. And now, if you go one step up, we will see that the solution of this is that Max prefers to take the right path, because he is of greater than equal to 5.

So, this side is 5, this side is 6. So, he prefers this side that brings him to 6. And then goes down this path which is 6. And there, goes down that path. And this is what Max does with the first best move that he has extracted from the minimax procedure and reduced search using alpha-beta pruning.

(Refer Slide Time: 39:08)



Effectiveness of Alpha-beta Pruning

- Alpha-beta is guaranteed to compute the same value for the root node as computed by minimax, with less or equal computation

Worst case:
- No pruning.
- Examining b^d leaf nodes, where each node has b children and a d-ply search is performed.

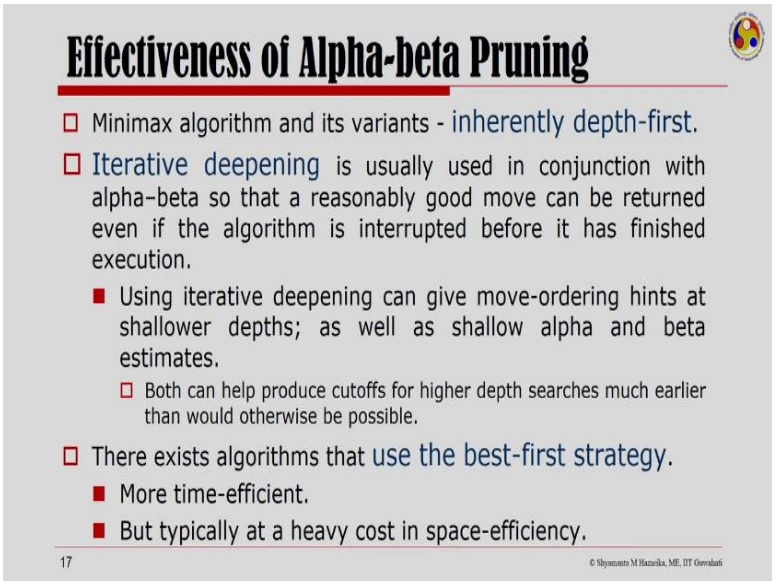
Best case:
- Examine only (2b)^(d/2) leaf nodes.
- Result is you can search twice as deep as minimax!

© Shyamanta M Hazarika, ME, IIT Guwahati

Now, alpha-beta pruning, the effectiveness is guaranteed by the fact that it is guaranteed to compute the same value for the root node, as computed by the minimax with less or equal computation. Worst case, if we do not have any pruning, we end up examining b to the power d leaf nodes, where each node had b children and we are talking of a d-ply search. In the best case, we examine only 2b to the power d by 2 leaf nodes.

What this result means is that, for minimax with alpha-beta pruning, we can search twice as deep as we would have done with minimax alone. So, let us say we were trying to get the first best move for a chess game and we were looking at a depth of 7 with minimax, then minimax with alpha-beta pruning, we can at least go up to 14. That is what the double means here.

(Refer Slide Time: 40:24)



Effectiveness of Alpha-beta Pruning

- Minimax algorithm and its variants - inherently depth-first.
- Iterative deepening is usually used in conjunction with alpha–beta so that a reasonably good move can be returned even if the algorithm is interrupted before it has finished execution.
  - Using iterative deepening can give move-ordering hints at shallower depths; as well as shallow alpha and beta estimates.
    - Both can help produce cutoffs for higher depth searches much earlier than would otherwise be possible.
- There exists algorithms that use the best-first strategy.
  - More time-efficient.
  - But typically at a heavy cost in space-efficiency.

17 © Shyamanta M Hazarika, ME, IIT Guwahati

Now, minimax algorithm and its variants are inherently depth-first search. So, there are improvements on minimax with alpha-beta pruning by using some better strategies for depth-first. Like, we could have iterative deepening, which is used in conjunction with alpha-beta pruning, so that a reasonable good move can be returned. Now, using iterative deepening can give move ordering hints at shallower depths and as well as shallow alpha and beta estimates.

Now, this can produce cutoffs for higher depth searches, much earlier than would otherwise be possible. There are other algorithms that use the best-first strategy for minimax with alpha-beta. Now, these are more time efficient. But then, typically it is heavy cost in terms of space recorded. These we have now covered in our game playing algorithms and leave it out as exercise to be taken during the course of our study. In the next week, we will focus on knowledge representation reasoning and understand predicate calculus. Thank you very much.